NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INPUT/OUTPUT FILTER PROGRAM FOR THE
WARFARE ENVIRONMENT SIMULATOR (WES)

by

Peter William Fotheringham

March 1982

Thesis Advisor: J.M. Wozencraft

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): An Input/Output Filter Program for the Warfare Environment Simulator (WES)	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis; March 1982
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Peter William Fotheringham	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California	12. REPORT DATE: March 1982 / 13. NUMBER OF PAGES: 95
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Warfare Environment Simulator
Naval Wargame
Wargame

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis describes the concept of an input and output filter for the Warfare Environment Simulator (WES). The filter is comprised of two computer programs which were written in the "C" programming language. The main program examines the input to the filter on a character by character basis, and provides two basic capabilities which are not currently available in WES. The first is the ability to edit or create files, and the second is a facility for translating

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

abbreviations (macros) into WES commands. The macros may include values which are substituted into variables in the commands as they are sent to WES. The second program buffers the output from WES as directed by the main program to avoid an interleaving of output when a process other than WES is in use.

Approved for public release; distribution unlimited.

An Input/Output Filter Program for the Warfare Environment Simulator (WES)

by

Peter William Fotheringham
Captain, United States Army
B.S., United States Military Academy, 1973

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(COMMAND, CONTROL AND COMMUNICATIONS)

from the

NAVAL POSTGRADUATE SCHOOL
March 1982

ABSTRACT

This thesis describes the concept of an input and output filter for the Warfare Environment Simulator (WES). The filter is comprised of two computer programs which were written in the "C" programming language. The main program examines the input to the filter on a character by character basis, and provides two basic capabilities which are not currently available in WES. The first is the ability to edit or create files, and the second is a facility for translating abbreviations (macros) into WES commands. The macros may include values which are substituted into variables in the commands as they are sent to WES. The second program buffers the output from WES as directed by the main program to avoid an interleaving of output when a process other than WES is in use.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I wish to gratefully acknowledge the guidance and encouragement of my thesis advisor, Professor Wozencraft, without whose assistance this thesis would not have been possible. Special appreciation is also due my second reader, Professor Lyons, whose time and advice were very much appreciated. Finally, I would like to thank the other students in the C3 Curriculum and the staff members of the Naval Postgraduate School who assisted me in designing and writing the computer programs.

I. BACKGROUND

A. OBJECTIVE

The objective of this thesis was to develop a "filter" program for the Warfare Environment Simulator (WES). It was envisioned that the program would continuously examine the characters which were being typed in to determine if a predetermined set of characters (a "string") was present, and if so, to perform some corresponding, and also predetermined function. Additionally, the program would be capable of buffering output from WES in the event that output from some other process is being displayed on the terminal. For example, this would allow for the use of an editor, since the output from WES could be saved, and later displayed on the terminal when the editing session was completed. This objective was met in the manner described in the following section.

B. INTRODUCTION

In achieving the stated objective, two major questions had to be answered. The first question was "What functions would be most useful in the implementation of the filter concept ?" Those who were most familiar with the operation of WES were the least concerned about functions which would make it easier to use: as usual, once a system has already

been learned, almost by definition its use becomes easy. On the other hand, since the majority of those who have tried to use WES have been initially overwhelmed by its complexity, they provided many suggestions as to what capabilities the filter program should provide. The second question was "How will the program determine when a particular function is to be performed ?," or similarly, "How will the user indicate that a function is to be performed ?" The answer to this question will be addressed first.

To enable a user of the program to distinguish between input to WES and a "command" to the filter program, a special character (the "$") was chosen. It must precede any input which is to be interpreted by the program. For example, if "help" was typed in, it would be passed through the filter to WES without any action being taken by the program, while "$help" would cause the program to display the help file without sending any characters to WES.

Answering the first question was a much more difficult task. The answer lies in the intersection of two lists of capabilities - a list of what would be most useful to implement, and a list of what is reasonable to implement. The adjective reasonable in this context means feasible in terms of the limitations imposed by: (1) the programming language being used ("C"), (2) the operating system (UNIX) and the standard routines it has access to, and (3) the time

available to write the program or subroutine. All of these factors were taken into account when the filter concept was developed and implemented.

A review of the capabilities which were common to the two lists resulted in a decision to include the following functions in the filter program:

1. An editing capability for reviewing old files, or creating new ones.

2. A means of identifying certain brief strings of input, and sending a much more complex series of characters to WES.

3. A mechanism for substituting arguments provided as input for variables in predefined WES commands.

4. A "default" mode of operation in which input provided to, as well as output from, WES occurs as it would if the filter program were not being used.

A detailed description of how these functions are performed is the topic of the next section.

II. PROGRAM STRUCTURE AND DESCRIPTION

A. THE MAIN PROGRAM

1. General

The main program receives input from the keyboard, checks for the special character, and either passes the other characters to telnet or responds to the "special" character strings as if they were commands. The special character for this program is the dollar sign ("$"). A "$" causes the program to compare the characters immediately following it to a list of strings of characters which correspond to subroutines which are to be performed. If the input string does not match a string in the list, an error message is generated and no action is taken.

These two modes of operation, as indicated in the source code, are (1) the "default" mode, and (2) the "call subroutine" mode. In the former, all input characters are sent to telnet, with the echo (a copy of the character on the terminal screen) being provided by the remote host. In the latter, the input characters are used to control the program, which performs various functions which are not otherwise available while running the WES program. These functions include listing the available files, invoking the editor, and expanding abbreviations (later referred to as macros) into standard WES commands.

The filter process was designed to be "file driven". For the most part, the text which is required for display in the main program is contained in separate files which are opened, read, and copied to the terminal. This allows for the text to be easily changed as necessary, and results in a shorter program. This tradeoff is in favor of both simplicity, and flexibility for future expansion or modification. It should be noted that this flexibility does not extend to changing the menu text without changing the switches in the main program which control the response of the program to the selection of an option.

Aside from the subroutines, the overall program is divided into three distinct parts (Figure 1). The main program controls the overall filter process. It also causes an asynchronous program to be executed which receives the output from telnet (which is the output from WES in this case), and either sends it directly to the terminal or stores it while other information is being displayed. This buffer program is discussed in more detail in section II C. The third part of the program is the telnet process, which will be discussed in section II D.

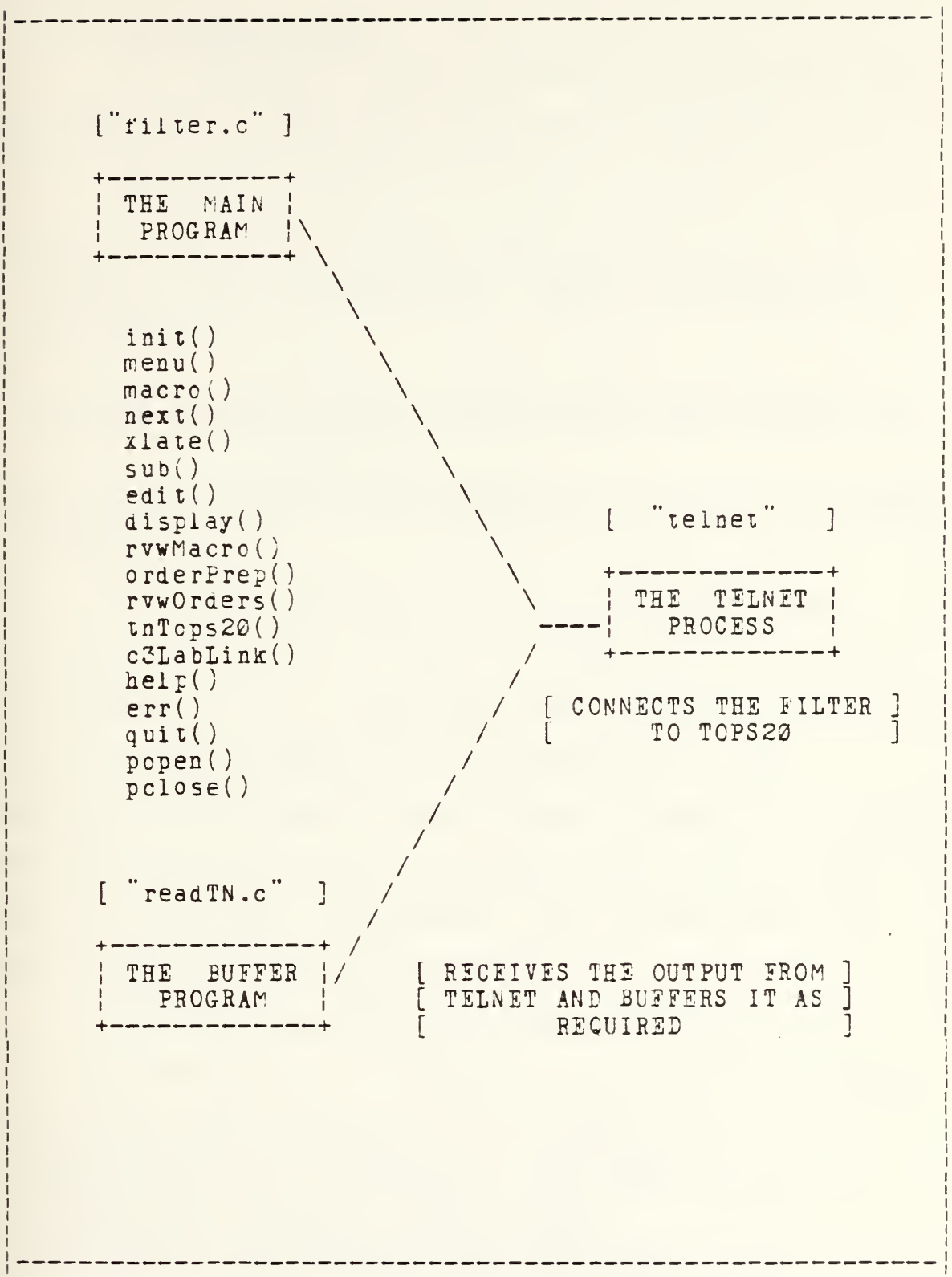


Figure 1. PROGRAM STRUCTURE

2. The Subroutines

The subroutine "init()" executes the actions which are required to begin using the program (Figure 2). The most significant of these actions are as follows:

1. Reading the macros and their WES equivalents into an array so that the program can rapidly determine if an input string is a valid macro, and if it is, output the WES command.

2. Determining the type of terminal that is being used, so that the proper (either line or full screen) editor can be called, if required.

3. Determining whether the program will be used with telnet, or for training on the capabilities and operation of the program.

The subroutine which controls the display and selection of options is "menu()". A text file containing the menu of options is displayed, and a number corresponding to the option selected is entered (Figure 3). The appropriate subroutine is then called. After an option has been completed, the menu is displayed again, and another choice may be made. Returning to the standard WES input mode is the last option on the menu.

The following options are available in the menu (with the exception of "4" which has not been implemented):

1. Editing an existing file, or creating a new file.
2. Displaying the current list of filenames.
3. Editing the list of macros and their WES equivalents.
4. Preparing an order for input to WES.
5. Displaying the current list of orders.
6. Returning to the standard WES input mode.

```
[ THE FILE "mac.text" IS OPENED AND READ INTO ]
[    THE ARRAY "mac[]" IN THE MAIN PROGRAM    ]


          Fri Mar 5 21:08:10 PST 1982

     WHAT TYPE OF TERMINAL ARE YOU WORKING ON ?

              1. Ann Arbor.
              2. Digital VT 100.
              3. A.D.M.3 or Tektronix.

            *** TYPE IN 1, 2 OR 3.***

1 <---[ A "1" WAS TYPED IN HERE TO INDICATE THAT ]
      [ THE TERMINAL BEING USED WAS AN ANN ARBOR ]

    WELCOME - You have the following options:

      1. Telnet to Tops20 and log in automatically.

      2. Link to another terminal in the C-3 Lab
         (primarily for training on this program).

            *** TYPE IN 1 OR 2.***


2 <---[ A "2" WAS TYPED IN HERE TO INDICATE THAT ]
      [   PROGRAM WAS TO BE USED FOR TRAINING    ]

      [ THIS RESULTS IN A CALL TO "c3LabLink()" ]
```

Figure 2. SUBROUTINE "init()"

```
        [ TYPING "$" AND A CARRIAGE RETURN CAUSES ]
        [   THE FOLLOWING MENU TO BE DISPLAYED    ]


    MENU #1 - You have the following options:

      1. Review/Edit an existing file.
      2. See the current list of filenames.

      3. Review/Edit the current macro list.

      4. Prepare an order.
      5. See the current list of orders.

      6. Quit (return to standard WES input mode).

         *** Type in 1, 2, 3, 4, 5, or 6. ***

6 <---[ A "6" TYPED HERE CAUSES THE PRCGRAM TO  ]
      [  RETURN TO THE STANDARD WES INPUT MODE  ]


      [ OPTION 4 HAS BEEN INCLUDED BOTH HERE AND ]
      [   IN THE SOURCE CODE TO DEMONSTRATE AN   ]
      [ OPTION WHICH MAY BE ADDED TO THE PROGRAM ]
      [  IN THE FUTURE, AND AS AN EXAMPLE OF HOW ]
      [    EASILY A MODULE MAY BE ADDED TO THE   ]
      [                 PROGRAM                  ]


@ <---[ THIS IS THE OPERATING SYSTEM PROMPT FROM ]
      [  TOPS20, WHICH IS DISPLAYED AFTER OPTION ]
      [          "6" HAS BEEN SELECTED           ]
```

Figure 3. SUBROUTINE "menu()"

Items 4 and 5 have been included in the menu as an example of a potential area for expanding the program beyond its capabilities as it is currently written. If option 4 (order preparation) is selected, a message is returned stating that the option is not implemented yet. Option 5, however, does list the files in the directory which end in ".ord" ; this can easily be changed to any ending which would readily identify the files containing standard groups of orders for input to WES. This will be discussed in more detail in Chapter IV.

The most complex series of subroutines begins with "macro()" (Figure 4). This family of routines includes "macro()", "next()", "xlate()", and "sub()". They are used to accomplish two closely related tasks, both of which are controlled by "macro()".

The first task is the substitution of a WES command, or a series of WES commands for an abbreviation (macro) which is defined and entered into the file "mac.text". This file may be changed at any time by choosing menu option 3. The file is stored in an array in the program to facilitate a rapid search and translate process. To expand a simple macro, a "$" is entered, followed by the macro and a carriage return. If the macro is to be used at the beginning (or in the middle) of a sentence, the macro is ended with the character "escape".

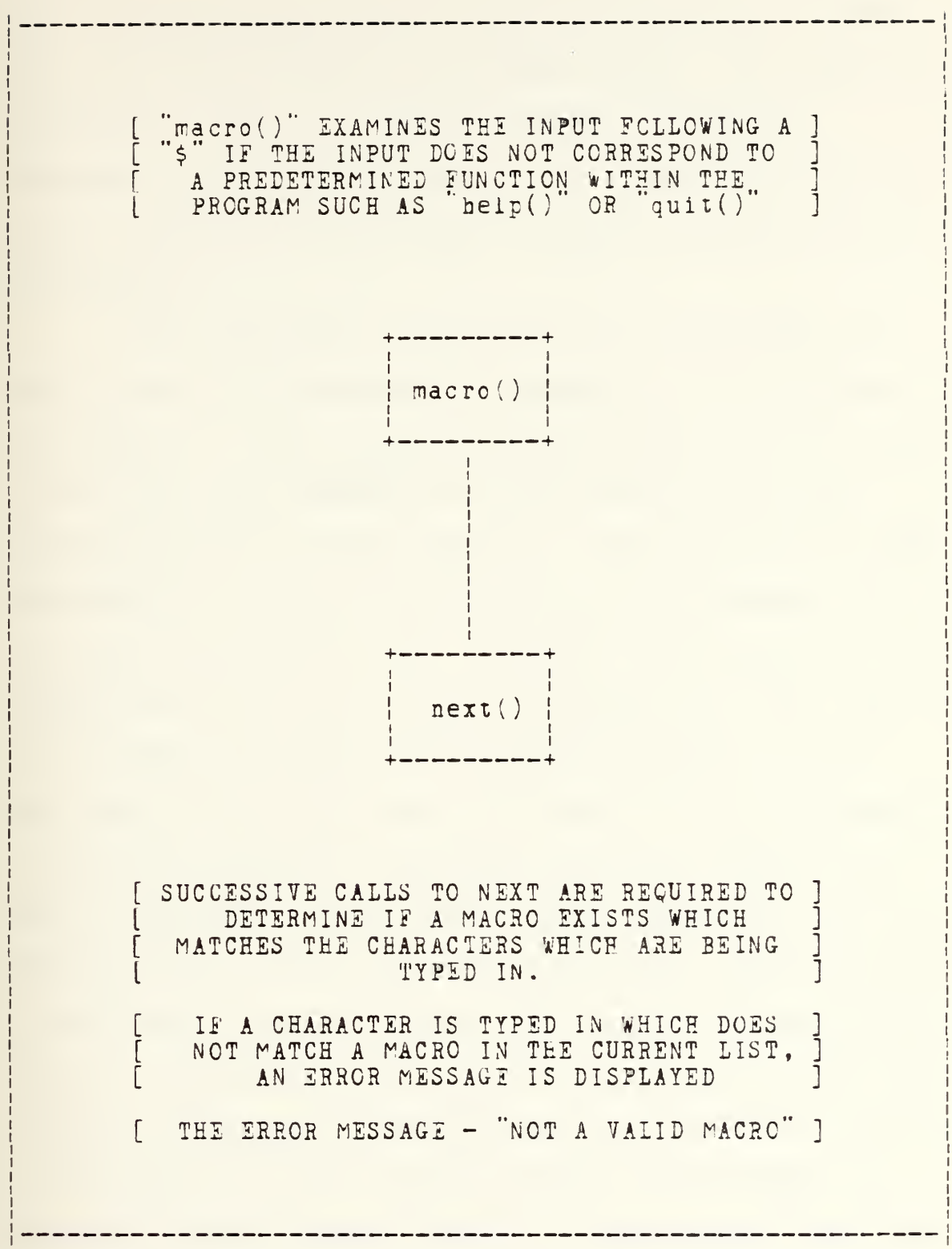


Figure 4. SUBROUTINES "macro()" AND "next()"

This causes the expansion of the macro to be sent to WES, but it is not followed by a carriage return. This type of expansion is useful for sending commands which are frequently used during the conduct of a game, especially since a new macro can be added to the file while the game is in progress.

The second task is an extension of the first in which a macro with arguments is expanded into one or more WES commands. After the macro and its expansion are found in the array, the arguments are substituted into their appropriate places in the commands as they are sent to WES. For example, if "$atk.a7(nimitz,4,flt1)" is entered, the following actions are taken. "Macro()" is called as soon as enough characters are entered to determine that the input is not a request for one of the programs standard routines, e.g. "help()". In this case, the letter "a" is sufficient to make this decision. Successive calls to "next()" are made as required to match the string of characters as it is being typed in. This serves two functions. First, no delay is required to locate the macro and send the WES equivalent. Second, as soon as a character is typed in which causes the string of characters to be different from the list of macros in "mac.text", an error message ("NOT A VALID MACRO") is displayed and the program returns to the standard WES input mode.

After "$atk.a7(nimitz,4,flt1)" is entered, it must be followed by a carriage return or an escape character. In either case, the arguments (nimitz, 4, and flt1) are substituted into the appropriate places in the command which is sent to WES. In this case, the output to WES would be "FOR NIMITZ LAUNCH 4 A7E FLT1 120 240 1000". Since this is a macro with arguments, the substitution and output to WES is accomplished by "sub()". If the macro did not have any arguments, the translation and output would have been accomplished by "xlate()" (Figure 5). As mentioned earlier, following a macro with a carriage return causes a carriage return to be sent to WES, while an escape character allows the macro expansion to be used in the context of a WES command .

"Edit()" is the routine which provides the capability to review an existing file or to create a new file. Since the type of terminal being used is determined in "init()", the corresponding type of editor can be called. The distinction between the different types of terminals must be made by the program, since the UNIX operating system is not capable of making the distinction on its own. After entering "edit()", the name of the file which is to be edited must be entered. Since the program examines every character which is typed in, if a mistake is made in typing in the file name, no provisions exist for deleting characters which have been typed incorrectly. To insure

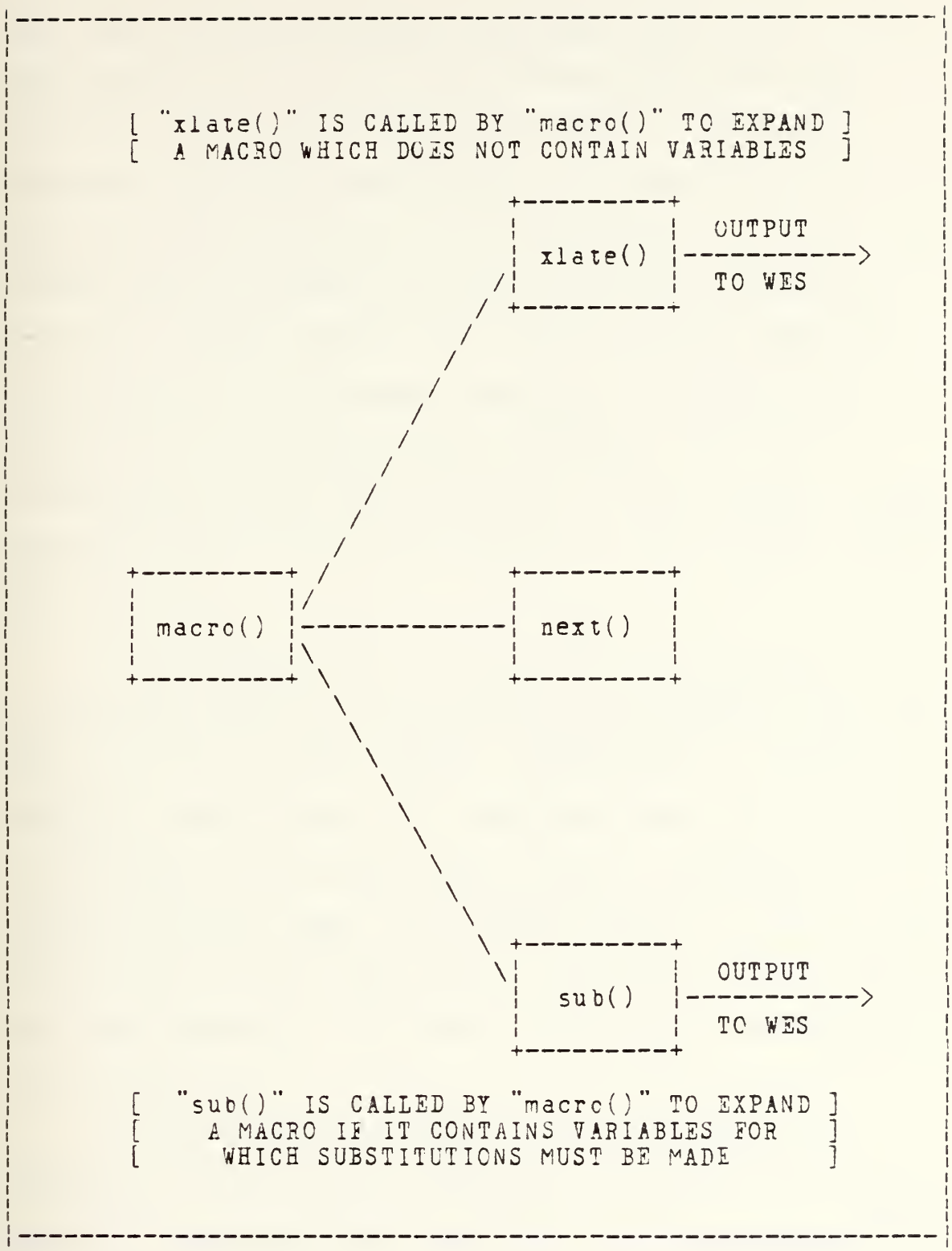


Figure 5. SUBROUTINES "xlate()" AND "sub()"

that this is not a problem, the program asks for a verification (a "y" or "n") of the name which has been entered. Once in the editor, the program is no longer examining the input, and the editor is used as it normally would be. As a minimum, this requires the user to know what command must be entered in order to exit from the editor, since the program cannot be of any assistance. Once the editing session is complete, the menu is displayed and another selection must be made. Although the file containing the macros may be edited in this fashion, a menu option has been provided to accomplish this directly (Figure 6).

The second option on the menu is to display the list of filenames which are available in the directory which was used to run the filter program. "Display()" is intended to provide a list of files if a particular file is to be edited or reviewed, but the exact name cannot be remembered. It presents the file names in a column format, so that all the files in most directories will be available on the screen at one time (Figure 6). To return to the menu, any character may be typed in.

As previously mentioned in the discussion on "edit()", the subroutine "rvwMacro()" automatically calls the macro expansion file ("mac.text") into the appropriate editor (Figure 6). "RvwMacro()" is called by selecting item 3 on the menu. This routine was included as an explicit

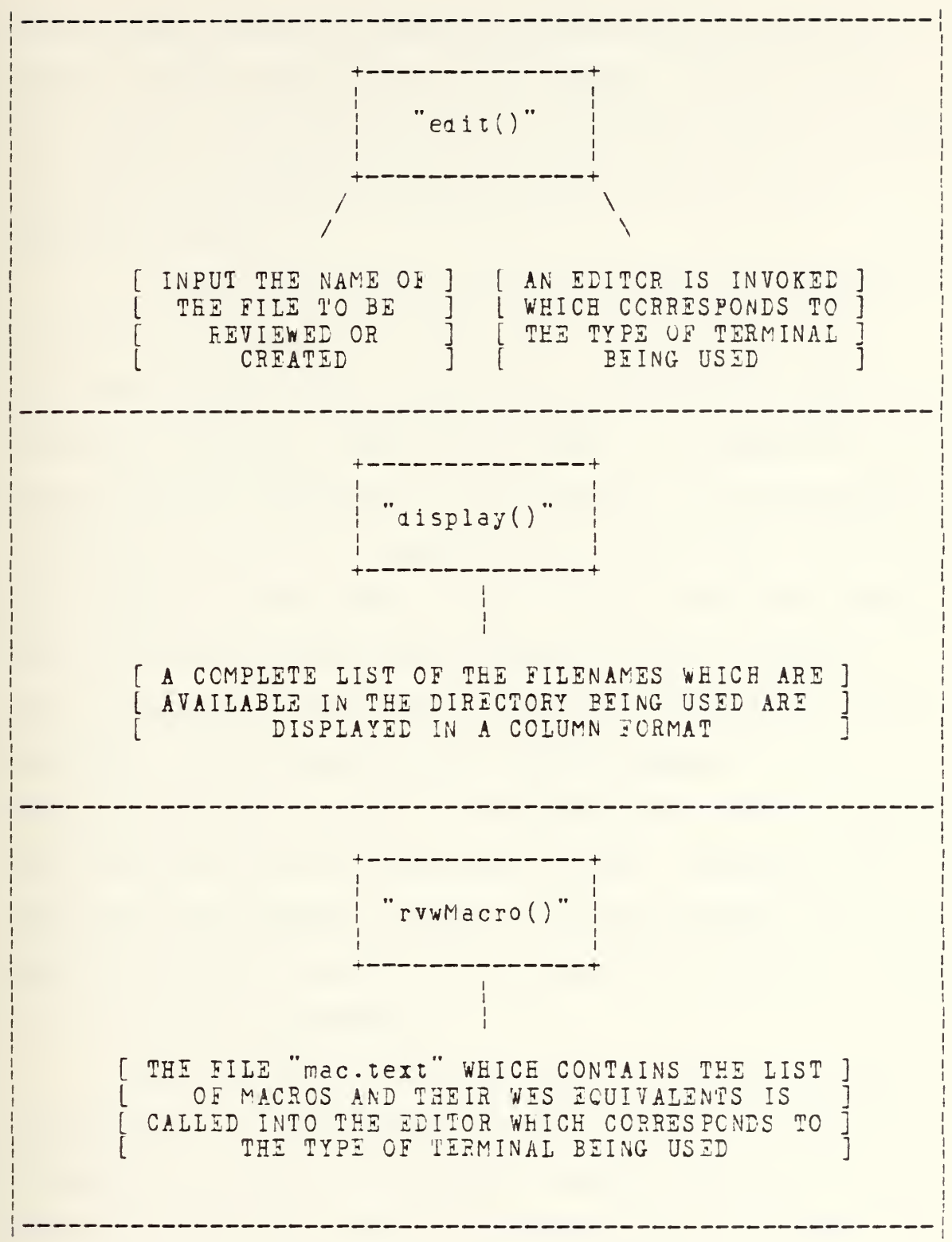


Figure 6. SUBROUTINES "edit()","display()" and "rvwMacro()"

option since the macro expansion capability is the most powerful option available in the program. In this light, this ability to rapidly add to or change the list of macros while the WES program is running should prove to be a desirable feature.

As mentioned earlier, the subroutine "orderPrep()" is not yet implemented, but has been included for the following reasons. The preparation of a series of WES commands, referred to here as an order, was suggested as a useful feature to be added to the program. In the context of WES such a series of commands would be called a "plan". This routine is envisioned to use a menu, from which small plans would be selected and combined to form a complete plan. The overall plan would contain standardized control characters, which would be replaced by arguments in a routine similar to "sub()". The plan could then be typed out or edited prior to being sent to WES. Although this description is somewhat limited, it should provide both an explanation of the concept and a framework on which to build if it is to be implemented.

The selection of the next item on the menu causes "rvwOrders()" to be executed. This routine is similar to "display()", in that it lists the available files of a certain type, in this case files which contain orders. These files are currently assumed to have a file type, or suffix, of ".ord". This suffix can be easily changed within

the program to one which is more easily remembered by the user. A series of suffixes can also be entered in the program if a more general listing of order files is desired.

The next subroutine is "tnTops20()", which performs a number of functions if the option to use telnet is chosen (Figure 7). First, it requests that two processes be initiated - telnet and "readTN". The output from telnet (WES) is sent to the background program, "readTN". This connection is established using the UNIX pipe function (|). However, the connection between the filter program and telnet is created in the program using "popen()". The pipe facility can not be used by the filter since a pipe sends all of the output of the first process to the second. For example, the pipe function would cause the menu to be sent to telnet, which obviously would not be desirable. "Popen()" creates a file descriptor, which is essentially a name for the input side of telnet. Using the file descriptor, selected output can be sent to telnet, while other information is displayed on the terminal screen.

As will be discussed more thoroughly in Section D of this chapter, "tnTops20()" also selectively adjusts the terminal modes, which would otherwise cause the keyboard to become inactive. It then sends the file "tnTops20.text" to telnet. The file contains the commands to log in and to begin running a certain program if desired. This is an example of why the program was designed to be file driven,

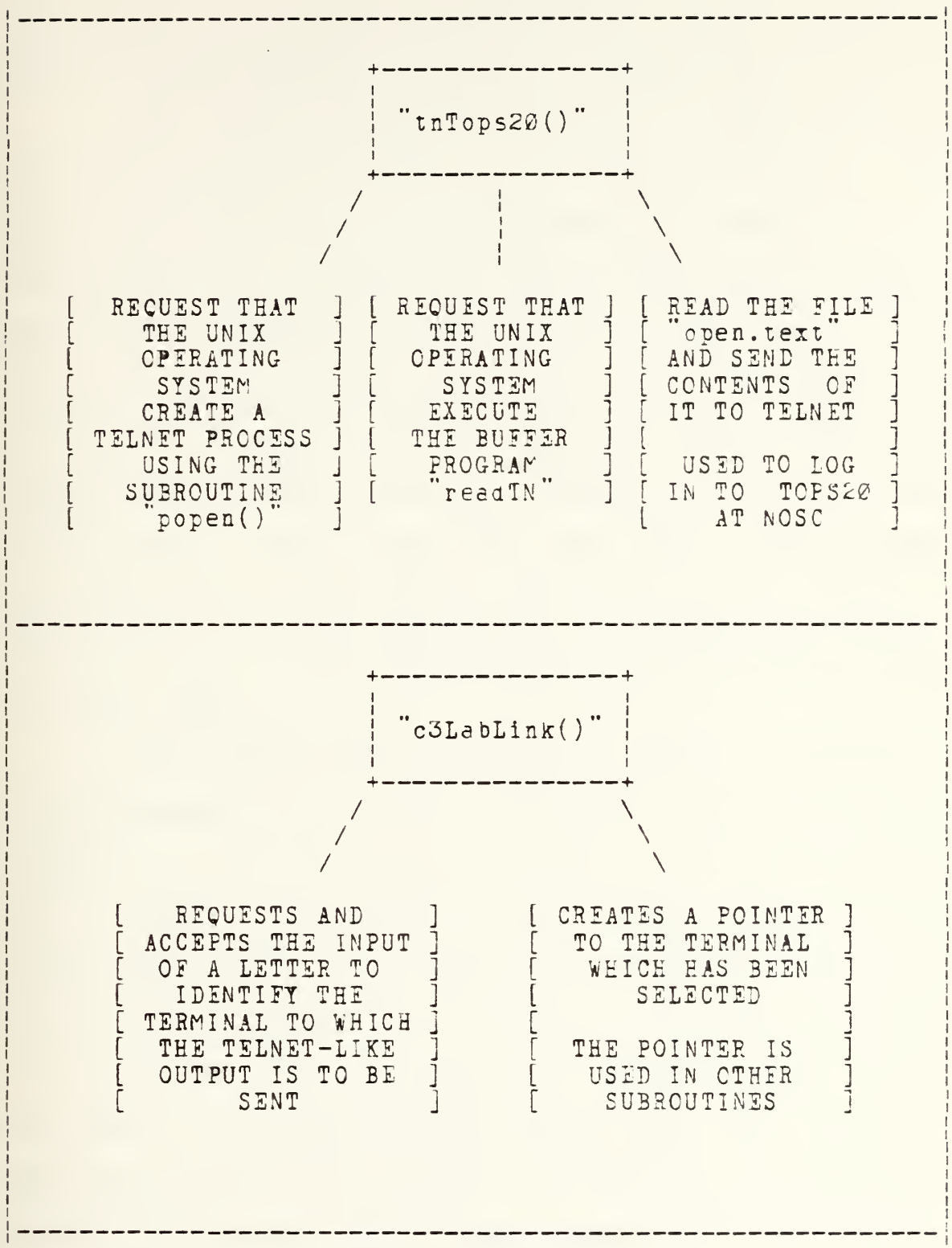


Figure 7. SUBROUTINES "tnTops20()" and "c3LabLink()"

for changing the program to be run after logging in is easily accomplished by editing the "tnTops20.text" file.

The "help()" routine is the first of the standard routines which are available in the program (Figure 8). It consists of a file which is displayed on the terminal and contains the basic formats for input to the filter program. The file is displayed by typing in "$ help" or "$?". The number of spaces occuring in the input string between words in this case is not significant, e.g., "$ help" and "$help" achieve the same result. However, this is not true for all input. For the most part, the input to the program is in the form of WES commands, which are handled by the "default mode" of the program, and as such are sent directly to telnet (WES).

The second standard routine is "quit()" (Figure 8). It is requested in the same manner as the help function, by typing in a dollar sign followed by the word quit ("$ quit"). Again, the number of spaces between the two words is not significant. This routine is used to exit from the program, after logging off of the system at NOSC. Logging off also causes the telnet process to be terminated, as indicated by a message on the terminal when this occurs. The filter program could simply be aborted at this point, however, the output buffer program would still be active, the terminal modes would not be set normally, and certain file descriptors would still be open. All the tasks

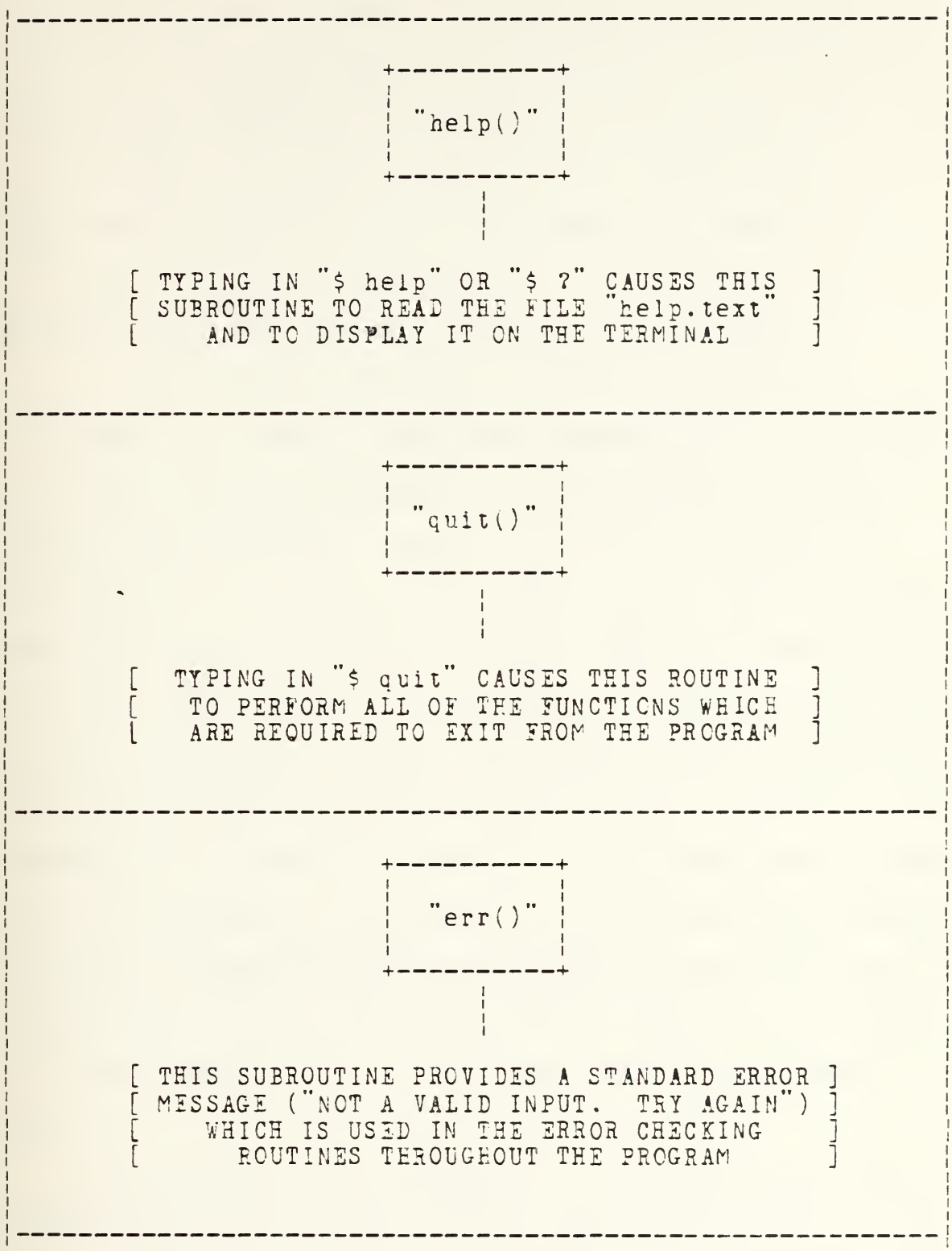


Figure 8. SUBROUTINES "help()", "quit()" AND "err()"

required to correct these deficiencies are accomplished by "quit()". A message indicating that the filter program is being exited properly is displayed as these tasks are accomplished. Failure to use the quit routine will, at best, result in the display of an operating system prompt ("%") with input from the keyboard being accepted, but not echoed (displayed) on the terminal as it is typed in.

As previously mentioned in the description of "tnTops20()", a useful connection between the filter program and telnet cannot be created using the UNIX pipe facility. Rather, the subroutine "popen()" must be used (Figure 9). The creation of a pipe in this manner allows the program to send only selected output to telnet. "Popen()" returns a file descriptor which, in this case, is essentially the name of the input side of telnet. The output which is intended for telnet is then directed using this file descriptor. The procedure for properly "closing" this file descriptor at the end of the program is discussed in the following paragraphs. Sufficient comments have been included in the source code to provide more details as to exactly how "popen()" functions. No further description of this routine will be included here.

A graceful (i.e., correct in a programming sense) termination of the filter program must include a function which closes the file descriptor ("fd") which is created in "popen()" (Figure 9). This action is accomplished in the

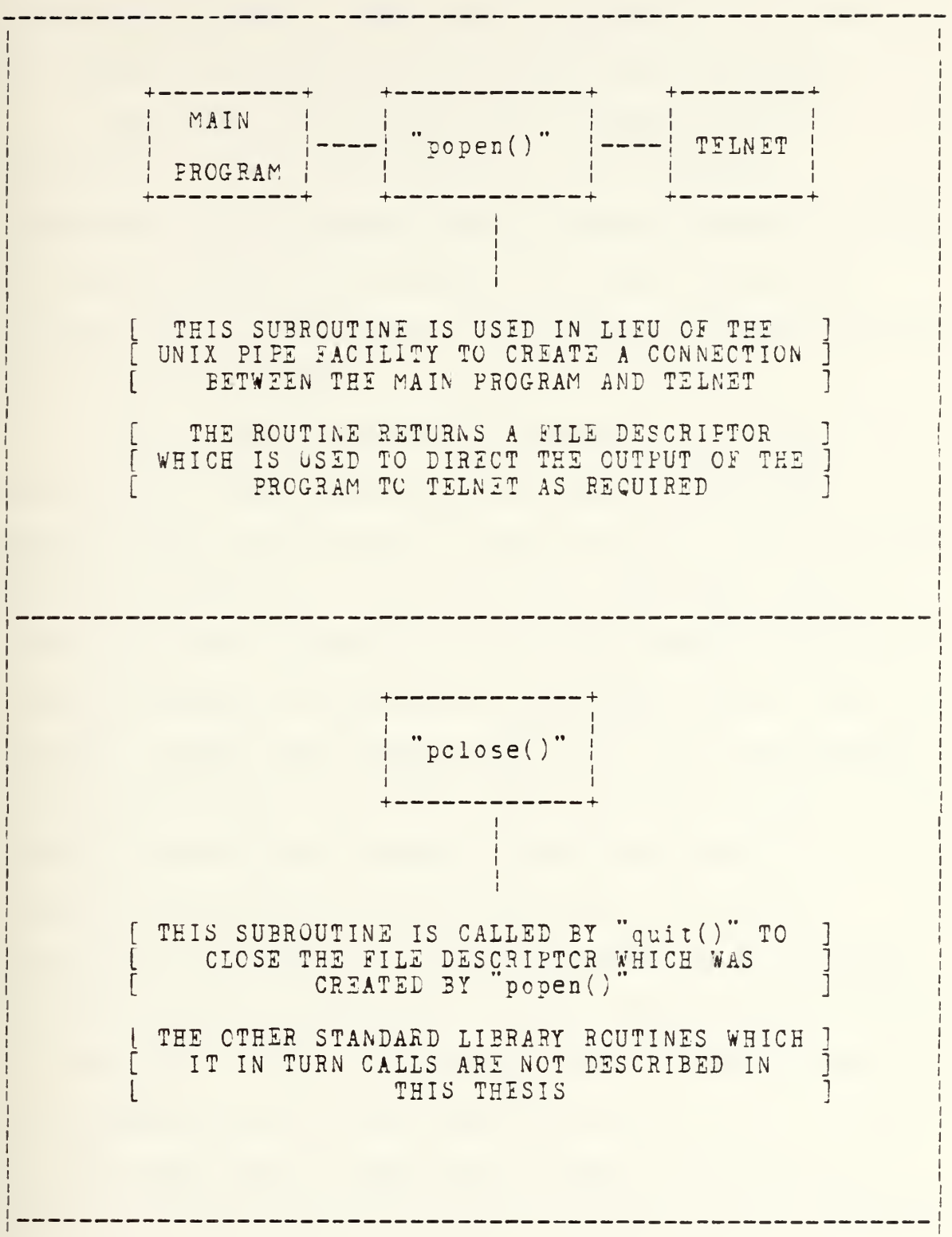


Figure 9. SUBROUTINES "popen()" AND "pclose()"

subroutine "pclose()". The call to "pclose()" is a part of the "quit()" routine which has already been discussed.

The final subroutine in the program is "err()" (Figure 8). It has been included to simplify and standardize the inclusion of error checking mechanisms which are found throughout the program. The only function it performs when it is called is the output of an error message on the terminal screen.

Due to the fact that the subroutines are file driven, all of the files which are required for their operation must be available in the directory from which the program is executed. If a file is not available, and an attempt is made to open it, an error will result which will stop the program. This type of catastrophic error could be avoided, however, the program would be at best handicapped by the absence of a particular file. For this reason, it is assumed that both the program and the files which are required to run it in its entirety will be available. Similarly, the "readTN" program must remain separate from the main program, and it too must be in the directory which is being used. A straightforward means of insuring that all the programs and files are accounted for is to store them all on a tape, and to read the tape when first using the filter, or when an error occurs because a required file has been changed or deleted.

B. THE UNIX INTERFACE

In general, the UNIX interface was the most straightforward concept to address in designing and implementing the filter program. In UNIX, essentially all areas where input and/or output can occur are treated as if they are files. To a great extent, this allows the user of a computer with a UNIX operating system to direct his attention to more important aspects of programming. This concept is also found in the "C" programming language, which is essentially the language of UNIX, and which is the language in which this program was written. In "C", however, a distinction is made between the standard input and output (which are generally the keyboard and terminal screen), a data file, and an active computer program (process), in that different formats must be used to receive input from or direct output to them. In most programs this would not be a major concern, but this program by design requires the use of all three formats.

As described earlier, the main program causes a second program to be executed. This program is called "readTN", and buffers the output from telnet (WES) as required. The readTN program runs as a background process, that is, it is executed by UNIX independently of the main program. It receives its instructions by having access to a file which contains a flag, i.e., a character whose value determines whether the buffer is to be active or not. The main program

also has access to the file containing the flag, and by changing the character in the file causes the buffer to be enabled ("1") or disabled ("2").

The main program also requests that the telnet process be initiated. This operation is handled by a call to the UNIX operating system, and will be discussed in more detail in the section on the telnet interface.

C. THE WES OUTPUT BUFFER PROGRAM

This program, "readTN", is essential to the operation of the filter process. Without a buffer, the output from WES would be mixed on the terminal screen with the output from the process which the user has selected. This would not have any effect on either WES or the user program, however, it would be at best confusing to the user. An additional, and perhaps more serious problem associated with interleaving the two outputs is that the user would be required to divide his attention between them. Buffering the WES output allows him to concentrate on one issue at a time, and to be quickly brought up to date when the second process is completed.

One means of controlling the output of the buffer is to use the X-ON/X-OFF feature which is a part of UNIX. In short, if the terminal page length ("pagelen") is not zero, and "flowout" is enabled, the UNIX operating system will send the output from WES or from the buffer program to the

terminal screen a page at a time. To continue with the next page, an X-ON, which is currently a "control-Q", is typed in. This character is interpreted by UNIX, but is not considered to be a valid input to the program. Similarly, a "control-S" can be used to stop the output before the end of a page is reached. Rather than displaying one page at a time, a "control-P" can be used to enter the "zoom mode", in which the output does not stop at the end of each page. The "control-P" is a toggle, that is, a second "control-P" causes the page length to once again take effect.

D. THE TELNET INTERFACE

Although not obvious in the source code, significant problems were encountered in implementing a program which provides input to the telnet process. The author of telnet, Dan Franklin of Bolt, Beranek, and Newman (BBN), stated that he did not anticipate the use of a program to "feed telnet" when he wrote the telnet process. He agreed that the input terminal becomes disabled when an attempt is made to create a connection between a process and telnet. This terminal "lockup" is caused by the telnet program, which changes the terminal modes in such a manner that no input can be generated from the keyboard. This only occurs when a program is used to create the telnet process. With this fact in mind, the program was written to reset the modes after the telnet process is created, so that the keyboard

will be reactivated. This is accomplished in the filter program in the subroutine "tnTops20()".

The main program creates telnet as a means of connecting the computer (a PDP 11/70) in the C3 Secure Computer Laboratory to one of the virtual machines at the Naval Ocean Systems Center (NOSC) in San Diego. Aside from the fact that a different operating system is being used after logging in to the system at NOSC, telnet itself is transparent to the user. It allows programs and processes to be run from the terminal as if a physical connection to the NOSC computer existed.

III. A USER'S GUIDE

The purpose of this section is to provide the user of the filter program with a concise, step-by-step set of instructions to follow in becoming familiar with the filter program. This is most easily accomplished by using the training mode, since the output of the program which would normally be sent via telnet to WES can be displayed on an adjacent terminal for easy reference. A copy of a training session on the filter program is found in Appendix A.

Three steps must be followed in order to begin training on the filter program. First, the type of terminal being used must be specified. This choice is made by simply entering the number from the menu which corresponds to the type of terminal from which the program is being run. Second, a "2" must be entered to use the program in the training mode. Finally, the letter which identifies the terminal to which the "telnet" output is to be sent must be entered. Although the letters which correspond to the Ann Arbor terminals are shown on the screen, and a list of all possible terminal identifiers is shown, this may not be adequate or simple enough to follow. As an alternative, "login" on the terminal which is to be used to display the output, as well as on the terminal to be used to run the filter program. On the keyboard of the display terminal,

type "dpy yourname", where "yourname" is the name used to login. This will show the upper or lowercase letter which corresponds to the terminal, or tty, immediately following the user name. Enter this letter when asked for the "tty you want to link to". Note that unlike many programs, the filter program reads every character as it is typed in, and a carriage return is not required after an entry is made unless it is specifically asked for, or after "$help", "$quit", or "$macro" is entered. Messages indicating that the link is being established, and that the keyboard can then be used will be displayed on the input terminal.

To review the three basic types of entries which can be made, enter "$ help", followed by a carriage return. It should be noted that this and any other entry which begins with a "$" must end with a carriage return (CR), or, in the case of macros, alternatively with an escape character. The three entries, "$ CR", "$ macro CR", and "$ quit CR" cause the menu, a macro expansion, and termination of the program to occur respectively.

In the training mode, a simulated system prompt ("@") is displayed on both the input and output terminals, since it would be seen if telnet was being used.

After reviewing the help file, input "$ CR" to display the menu of options. Step through each of the six items on the menu, returning to the standard WES input mode by choosing the sixth option. At this time, the macro

expansion capability can be demonstrated. If a copy of the file "mac.text" is not available, return to the menu, edit the macro list by selecting item 3, and write down one or more macros. Enter a "$" followed by the macro (with arguments, if any) and end the input with a carriage return. If the macro has been entered properly, no error message will be displayed, and the WES command(s) will be displayed on both the input and the output terminals. Insure that the proper number of arguments, in the proper sequence, are entered.

For easy reference, abbreviations for the arguments have been included in each macro in "mac.text". For example, "s" must be replaced by a ship name, "n" by a flight name, and "#" by the quantity of items (e.g. F-14's) which are desired. It is anticipated that a paper copy of the macro file will be helpful and should be available if a relatively large number of macros are being used. The addition of the letters to represent the arguments of a macro should also be accomplished when a new macro is added to the list, to insure that it is properly documented for future use.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The input and output filter program has been successfully demonstrated to be a viable concept which interfaces well with both the Warfare Environment Simulator (WES) and the user. At the present time, the program has been tested with WES being run using several different, but predetermined scenarios. This approach has been sufficient to detect and correct deficiencies in the programs which make up the filter, but a recommendation for future testing and use is made in the next section.

B. RECOMMENDATIONS

1. Future Testing

Although the program was tested on an incremental basis as it was developed, and significant changes were made to insure that its use did not interfere with the conduct of a WES wargame in any way, more extensive testing should be accomplished as follows. The program should be used during a truly interactive wargame, as opposed to the wargames which have recently been run in which essentially all orders were entered from a prepared script. An interactive wargame would be especially useful in evaluating the actual and potential value of the most powerful capability which the

program provides, i.e. macro expansion. Any scenario in which the player is forced to be active a great deal of the time would be most useful in accomplishing this evaluation.

2. Other Applications

A general observation and recommendation for further work stems from the fact that the input/output filter concept has an unlimited number of possible applications. In general, interactive computer programs such as WES are readily accessible, but not flexible or easy to modify, if they can be changed at all. The filter concept, on the other hand, provides a mechanism for achieving a wide range of flexibility in terms of aids and options which can be provided while remaining within the constraints on input imposed by an otherwise inflexible process. The general structure which has been presented here should serve as an adequate baseline from which to develop many variations of the filter program.

APPENDIX A

SAMPLE OUTPUT FROM THE FILTER PROGRAM

The attached text is a copy of the actual output of the filter program when used in the training mode. The only significant difference between the output shown here and the output when the program is used with telnet is as follows. In telnet, the echo is provided by the remote host (TOPS20), so that both the input echo and the output from WES appear in capital letters.

The copies of the menu of options which occur after it is displayed the first time have been removed and replaced with "(MENU HERE)".

Mon Mar 5 20:57:02 PST 1982

WHAT TYPE OF TERMINAL ARE YOU WORKING ON ?

1. Ann Arbor.
2. Digital VT 100.
3. A.D.M.3 or Tektronix.

*** TYPE IN 1, 2 OR 3.***

3 (SELECT OPTION #3)

WELCOME - You have the following options:

1. Telnet to Tops20 and log in automatically.
2. Link to another terminal in the C-3 Lab
   (primarily for training on this program).

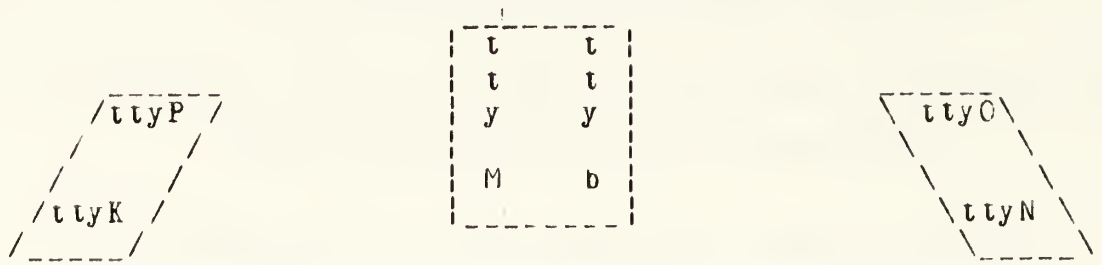
*** TYPE IN 1 OR 2.***

2 (SELECT OPTION #2)

Choose one of the following devices to link to :

tty	ttyB	ttyA	ttyB	ttyC
ttyD	ttyE	ttyF	ttyG	ttyH
ttyJ	ttyK	ttyL	ttyM	ttyN
ttyO	ttyP	ttyQ	ttyR	ttyS
ttyT	ttyU	ttyV	ttyW	ttyX
ttyY	ttya	ttyb	ttyc	ttyd
ttye	ttyf	ttyg	ttyh	ttyi
ttyj	ttyk	ttyl	ttym	ttyn
ttyo	ttyp	ttyw	ttyy	ttyz

The current tty layout in the lab is :
(Ann Arbor Terminals)



*** TYPE IN THE tty YOU WANT TO LINK TO ***
*** FOLLOWED BY A CARRIAGE RETURN ***
*** FOR EXAMPLE ----> ttyL ***

----> ttyQ

LINK TO ttyQ
IS THIS CORRECT ? (TYPE y OR n) --> y
LINKING TO C-3 LAB TERMINAL ttyQ.
RETURNING CONTROL OF THE KEYBOARD TO YOU.

@$help[CARRIAGE RETURN]

"$ help" or "$?" causes this file to be displayed.

The "$" character is used to enter the portion of the program which is NOT transparent to you. The following syntax must be followed:

1. "$ CR" ($ followed by a carriage return) -

   Used to enter the 'Menu of Options' mode.

2. "$ macro CR" ($ followed by a macro AND a carriage return) - (OR an escape character)

   Used to expand a macro into a WES command (or into a series of WES commands).

3. "$ quit CR" ($ followed by quit AND a carriage return) -

   Used to exit the program gracefully.

*** RETURNING TO THE STANDARD WES INPUT MODE.***

```
@$[CARRIAGE RETURN]

     MENU #1 - You have the following options:

       1. Review/Edit an existing file.
       2. See the current list of filenames.

       3. Review/Edit the current macro list.

       4. Prepare an order.
       5. See the current list of orders.

       6. Quit (return to standard WES input mode).

          *** Type in 1, 2, 3, 4, 5, or 6. ***

1 (SELECT OPTION #1)

       TYPE IN THE NAME OF THE FILE YOU WANT TO EDIT,
               FOLLOWED BY A CARRIAGE RETURN ---> menu.text

       EDIT FILE  menu.text

       IS THIS CORRECT ? (TYPE y OR n) --> y


                    edn menu.text

                GOING TO THE EDITOR NOW.
```

```
16>1,16[CARRIAGE RETURN]  (TO LIST LINES 1-16)
 1
 2
 3        MENU #1 - You have the following options:
 4
 5      1. Review/Edit an existing file.
 6      2. See the current list of filenames.
 7
 8      3. Review/Edit the current macro list.
 9
10      4. Prepare an order.
11      5. See the current list of orders.
12
13      6. Quit (return to standard WES input mode).
14
15
16         *** Type in 1, 2, 3, 4, 5, or 6. ***
16>q[CARRIAGE RETURN]  (TO EXIT FROM EDN)
```

(MENU HERE)

2 (SELECT OPTION #2)

DISPLAYING THE CURRENT LIST OF FILENAMES.

abs	appen	appen~	blue.dir	buffer.c
buffer.text	c.compiler	c3	c3.doc	c3LL.text
cal.c	cark.c	ch.c	clark.c	clark2.c
clear.c	cntrlf.dir	control.dir	cwc.dir	db
fig2	filter	filter.c	filter.c~	flag1
flag2	flagTN	flagTNout	force	garb
itx	larry	loop	ltr	m.c
mac	mac.text	mac.text~	menu.text	menu1
menu1.c	menu1.o	menu2.text	new	new.c~
new.pg	one.c	open.text	orderPrep.text	page
q	r	r2.c	readTN	readTN.c
red.dir	rem.c	rvw.text	r~	s2.c

TYPE ANY CHARACTER TO RETURN TO THE MENU --> [CARRIAGE RETURN]

(MENU HERE)

```
3 (SELECT OPTION #3)

       EDITING THE MACRO EXPANSION FILE NOW.

1078
27>1,27
 1 MACRO LIST           STRING TO BE SUBSTITUTED FOR THE MACRO
 2 ----------           --------------------------------------
 3
 4 cap(s,n)$            ENTER PLAN C[B
 5                      FOR [A LAUNCH 4 F14 [B 030 240 28000
 6                      LOAD 4 PHENX 4 SWDR
 7                      MISSION CAP
 8                      STOP $
 9
10 atk.a7(s,#,n)$       FOR [A LAUNCH [B A7E [C 120 240 1000 $
11 radar(s)$            FOR [A ACTIVATE RADAR $
12 macro4$              submac4 $
13 macro5$              submac5 $
14 macro6$              submac6 $       ( A "[" HERE INDICATES A  )
15 macro7$              submac7 $       ( CONTROL CHARACTER WHICH )
16 macro8$              submac8 $       ( OTHERWISE COULD NOT BE  )
17 macro9$              submac9 $       (          PRINTED        )
18 macro10$             submac10 $
19 mac19()$             Firstword is [A. $
20 mac20()$             Firstword [A, secondword [B, firstagain [A.$
21
22
23       THE LAST CHARACTER OF BOTH THE MACRO AND ITS EXPANSION
24           MUST BE THE DOLLAR SIGN CHARACTER "$"
25           ==== ==
26
27         s = ship     # = quantity     n = flight name
27>q[CARRIAGE RETURN]   (TO EXIT FROM EDN)
```

(MENU HERE)

4 (SELECT OPTION #4)

ORDER PREPARATION - NOT IMPLEMENTED YET.

(MENU HERE)

5 (SELECT OPTION #5)

THE FOLLOWING ARE THE CURRENT ORDER FILES :

test.ord

(MENU HERE)

6 (SELECT OPTION #6)

RETURNING TO THE STANDARD WES INPUT MODE.

```
@$cap(nimitz,air1)[CARRIAGE RETURN]  (MACRO EXPANSION)

        SENDING --> ENTER PLAN Cair1
                    FOR nimitz LAUNCH 4 F14 air1 030 240 28000
                    LOAD 4 PHENX 4 SWDR
                    MISSION CAP
                    STOP

        RETURNING TO THE STANDARD WES INPUT MODE.

@$atk.a7(halsey,4,atk2)[CARRIAGE RETURN]  (MACRO EXPANSION)

        SENDING --> FOR halsey LAUNCH 4 A7E atk2 120 240 1000

        RETURNING TO THE STANDARD WES INPUT MODE.

@$mac20(one,two)[CARRIAGE RETURN]  (MACRO EXPANSION)

        SENDING --> Firstword one, secondword two, firstagain one.

        RETURNING TO THE STANDARD WES INPUT MODE.

@$quit[CARRIAGE RETURN]

        THE PROGRAM IS TERMINATING GRACEFULLY.

%  (THIS IS THE UNIX SYSTEM PROMPT)
```

APPENDIX B

THE MAIN PROGRAM

```
#
#include <stdio.h>
#include <signal.h>
#define MACSIZE  6400
#define SUBMAC   800
#define WRITE    1
#define READ     0
#define EOF      -1

            /*    THIS IS THE WES "FILTER" PROGRAM
                  COMPILE THIS USING "pcc filter.c -lS" */

char linkdevice[] {'/','d','e','v','/','t','t','y','M','/','\0'} ,
     d, e,                /* Input characters used in switches.....*/
     CR 012,              /* The octal equivalent of "carriage return"....*/
     device[12],          /* The terminal to link to for training.....*/
     buffer[81],          /* The "macro" to be expanded and sent to WES...*/
     mac[MACSIZE + 2] ;   /* The array of macros and their equivalents....*/
int  flagN,               /* Equal to 0 when a macro MAY be found.....*/
     flagCR,              /* Equal to 0 when macro NOT ended by CR.....*/
     popen(), pclose(),   /* Two subroutines for opening/closing a pipe...*/
     popen_pid,           /* Common to popen() and pclose().....*/
     i,j,k,               /* Counters for loops.....*/
     option,              /* Option chosen - Telnet or Training.....*/
     termtype,            /* Terminal type - "Smart" or "Dumb".....*/
     fout,                /* The file descriptor for output to telnet.....*/
     mode ;               /* The mode (read or write) for popen().....*/
FILE *fpTN,               /* The file descriptor for output to a tty.....*/
     *FP ;                /* The pointer to the buffer on-off flag.....*/
```

```
main()  /* The main program begins here ..............................*/
        {
        char c, f, g, h ;
        extern char d, e ;
        int flag2 ;     /* To choose between help() & quit()..........*/
        extern int option ;
        extern FILE *FP, *fpTN ;

        init() ;        /* Actions required to begin the program.......*/

/* The default mode of operation begins here.........................*/

        if ( option == 1 )
                {
                FP = fopen("flagTN", "w") ;
                putc('2', FP) ;          /* Receive WES output.........*/
                fclose(FP) ;
                }

      labelA:
        c = getchar() ;

        switch(c)
                {
                case '$' :      /* Dollar sign........................*/

                        if ( option == 1 )
                                {
                                FP = fopen("flagTN", "w") ;
                                putc('1', FP) ; /* Buffer WES output....*/
                                fclose(FP) ;
                                }

                        goto labelB ;
                        break ;
```

```
                default :       /* All other characters.................*/
                        fpTN = fopen(linkdevice, "w") ;
                        if ( option == 1 )      /* Pipe is on here......*/
                                fdprintf(fout,"%c",c) ;
                        if ( option == 2 )
                                putc(c, fpTN) ;
                        if ( option == 2 && ((c == '\n') || (c == 015)))
                                {
                                putc('@', fpTN) ; /* To simulate the ...*/
                                printf("@") ;     /*  system prompt  ...*/
                                }
                        fclose(fpTN) ;
                        goto labelA ;
                        break ;
                }

/* End default mode.*/          /*  Begin call subroutine mode.............*/

        labelB:
          d = getchar() ;
          switch(d)
                        {
                        case '\n' : menu() ;
                                goto labelA ;
                                break;
                        case 015 : menu() ;
                                goto labelA ;
                                break;
                        case ' ' : goto labelB ;
                                break ;
                        case '?' : help() ;
                                goto labelA ;
                                break ;
```

```
        case '$' : err() ;
                   goto labelA ;
                   break ;
        case 'h' : e = getchar() ;
                   break ;
        case 'H' : e = getchar() ;
                   break ;
        case 'q' : e = getchar() ;
                   break ;
        case 'Q' : e = getchar() ;
                   break ;
        default :
                   if ( option == 1 )
                           putchar(d) ;
                   macro() ;
                   goto labelA ;
                   break ;
        }
switch(e)
        {
        case 'e' : f = getchar() ;
                   break ;
        case 'E' : f = getchar() ;
                   break ;
        case 'u' : f = getchar() ;
                   break ;
        case 'U' : f = getchar() ;
                   break ;
        default : err() ;
                   goto labelB ;
                   break ;
        }
```

```
switch(f)
     {
     case 'l' : g = getchar() ;
                break ;
     case 'L' : g = getchar() ;
                break ;
     case 'i' : g = getchar() ;
                break ;
     case 'I' : g = getchar() ;
                break ;
     default : err() ;
               goto labelB ;
               break ;
     }
        switch(g)
               {
               case 'p' : h = getchar() ;
                          flag2 = 0 ;
                          break ;
               case 'P' : h = getchar() ;
                          flag2 = 0 ;
                          break ;
               case 't' : h = getchar() ;
                          flag2 = 1 ;
                          break ;
               case 'T' : h = getchar() ;
                          flag2 = 1 ;
                          break ;
               default : err() ;
                         goto labelB ;
                         break ;
               }
```

```
                                        switch(h)
                                             {
                                             case '\n' :
                                                     if(flag2 == 0)
                                                       help() ;
                                                     if(flag2 == 1)
                                                       quit() ;
                                                     goto labelA ;
                                                     break ;
                                             case 015  :
                                                     if(flag2 == 0)
                                                       help() ;
                                                     if(flag2 == 1)
                                                       quit() ;
                                                     goto labelA ;
                                                     break ;
                                             default :
                                                     err() ;
                                                     goto labelA ;
                                                     break ;
                                             }

}       /* The main program ends here.....................................*/
```

```
/* SUBROUTINES */

init()  /* Actions required to begin the program..................... */
        {
        char m1 ;
        extern char device[12] ;
        extern int option ;
        FILE *fp3, *fp6 ;

        for ( i = 0; i <= 10; ++i )
                device[i] = ' ' ;       /* Clear string "device"........*/

        device[0]  = '/' ;
        device[1]  = 'd' ;
        device[2]  = 'e' ;
        device[3]  = 'v' ;
        device[4]  = '/' ;          /* "/dev/ttyM/" is the default .......*/
        device[5]  = 't' ;          /*   value of the terminal to  .......*/
        device[6]  = 't' ;          /*    link to for training.    .......*/
        device[7]  = 'y' ;
        device[8]  = 'M' ;
        device[9]  = '/' ;
        device[10] = '\0' ;

        fp3 = fopen("mac.text", "r") ;  /* Read the macro list and the .*/
        i = 1 ;                         /* list of substitutions into  .*/
        while ((m1 = getc(fp3)) != EOF) /*      the array "mac".       .*/
                {
                mac[i] = m1 ;
                i = i + 1 ;
                }
        fclose(fp3) ;

        printf("%c %c \t\t",CR,CR) ;
        system("date") ;
```

```
    printf("\n\tWHAT TYPE OF TERMINAL ARE YOU WORKING ON ?") ;
    printf("\n\n\t\t1. Ann Arbor.") ;
    printf("\n\t\t2. Digital VT 100.") ;
    printf("\n\t\t3. A.D.M.3 or Tektronix.") ;
    printf("\n\n\t\t*** TYPE IN 1, 2 OR 3.***\n") ;

    system("rodtty brkall") ;
label5 :
    m1 = getchar() ;          /* Read in the type of terminal.........*/
    switch(m1)
            {
            case '1' :
                    termtype = 1 ;  /* Used in edit() to determine .*/
                    break ;         /* what type of editor to use  .*/
            case '2' :
                    termtype = 2 ;
                    break ;
            case '3' :
                    termtype = 3 ;
                    break ;
            default :
                    err() ;
                    goto label5 ;
            }
label3 :

    fp6 = fopen("open.text", "r") ;
    while(( m1 = getc(fp6)) != EOF )
            putchar(m1) ;
    fclose(fp6) ;
```

```
        m1 = getchar() ;
        switch(m1)    /* Selection from opening menu..................... */
                {
                case '1' :
                        option = 1 ;
                        tnTops20() ;     /* To use the program with WFS..*/
                        break ;
                case '2' :
                        option = 2 ;
                        c3LabLink() ;    /* To use it for training only..*/
                        break ;
                default :
                        err() ;
                        goto label3 ;
                        break ;
                }

        return ;
        }


menu() /* The menu of options... ..................................... */
        {
        char m2 ;
        extern int fout ;
        extern int option ;
        FILE *fp ;
        extern FIIE *fpTN ;

      label0:
        if ( option == 1 )
                system("modtty echo") ;
```

```
fp = fopen("menu.text", "r") ;
while ((m2 = getc(fp)) != EOF)
        putchar(m2) ;
fclose(fp) ;
system("rodtty brkall") ;
m2 = getchar() ;
switch(m2)    /* Selection from the menu.....................*/
        {
        case '1' :
                edit() ;
                break ;
        case '2' :
                display() ;
                break ;
        case '3' :
                rvwMacro() ;
                break ;
        case '4' :
                orderPrep() ;
                break ;
        case '5' :
                rvwOrders() ;
                break ;
        case '6' :
                printf("%c%c\tRETURNING TO THE STANDARD ",CR,CR) ;
                printf("WES INPUT MODE.%c",CR) ;
                fdprintf(fout,"%c",CR) ;   /* Pipe is ON here...*/
                if ( option == 2 )
                        {
                        fpTN = fopen(linkdevice, "w") ;
                        putc(CR, fpTN) ;
                        putc('@', fpTN) ; /* To simulate the ...*/
                        printf("@") ;     /*  system prompt  ...*/
                        fclose(fpTN) ;
                        }
```

```
                        break ;   /* = return in this case.............*/
                default :
                        err() ;
                        goto label1C ;
                        break ;
                }

        if ((m2 == '1') || (m2 == '2') || (m2 == '3') || (m2 == '4')
                        || (m2 == '5'))
                goto label1C ;

        if ( option == 1 )
                {
                system("modtty -echo") ;

                FP = fopen("flagTN", "w") ;
                putc('2', FP) ;         /* Receive WES output.....*/
                fclose(FP) ;

                fdprintf(fout,"\n") ;
                }

        return ;
        }       /* End of menu of options.............................*/

macro() /* Check to see if a given macro is valid. ....................*/
        {
        int flag ;
        extern int fout ;
        extern int flagN ;
        extern int flagCR ;
        extern int option ;
        extern char d ;
        extern char buffer[81] ;
        extern FILE *FP ;
```

```
labelD:
  flag = Ø ;        /* flag = Ø   when NOT a valid macro.     .......*/
  flagN = Ø ;       /* flagN = Ø  when macro MAY be found.    .......*/
  flagCR = Ø ;      /* flagCR = ℓ when macro not ended by CR  .......*/
  i = -1 ;
labelF:
  if ( i <= MACSIZE + 1)
          i = i + 1 ;
  j =  i  ;                 /* "j" is the # of one of the     .......*/
  if ( j >= MACSIZE )       /*    characters in array mac     .......*/
          goto labelG ;
  if ( d != mac[j] )        /* Drops out of this loop with    .......*/
          goto labelF ;     /*  j = the index of the FIRST    .......*/
  if ( d == mac[j] )        /*  string which begins with the  .......*/
          ; /* null*/      /*  value of the variable "d"     .......*/
  buffer[1] = d ;
  for ( k = 1; k < 20; k = k + 1 )
          {
          e = getchar() ;

          if ( option == 1 )
                  putchar(e) ;

          buffer[k + 1] = e ;
          if (( mac[j + k] == e ) && ( mac[j + k + 1] == '$'))
                  {
                  xlate() ;  /* Output the WES equivalent........ */
                  if ( flagCR == 1 )
                          {
                          flag = 1 ;
                          break ;
                          }
                  else
                          ;  /* Null statement ..................*/
                  }
```

```
if (( mac[j + k] == e ) && ( mac[j + k + 1] == '(' ))
        {
        sub() ;  /* Create the WES equivalent..... */
        if ( flagCR == 2 )
                {
                flag = 1 ;
                break ;
                }
        else
                ;  /* Null statement ..................*/
        }

if ( mac[j + k] == e )
        {
        continue ;
        }

if ( mac[j + k] != e )
        {
        next() ;              /* Find the next match......*/
        if ( flagN == 1 )     /* Macro not found.........*/
                break ;
        if ( mac[j + k + 1] == '$' )
                {
                xlate() ; /* Output the WES equivalent..*/
                if ( flagCR == 1 )
                        {
                        flag = 1 ;
                        break ;
                        }
                else
                        continue ;
                }
```

```
                        else if ( mac[j + k + 1] == '(' )
                                {
                                sub() ; /* Create the WES equivalent....*/
                                if ( flagCR == 2 )
                                        {
                                        flag = 1 ;
                                        break ;
                                        }
                                else
                                        continue ;
                                }
                        else
                                continue ;
                        }
                }

        labelG:
          if (flag == 0)  /* It is NOT a valid macro.....*/
                {
                printf("%c \tNOT A VALID MACRO. %c",CR,CR) ;
                printf("%c %c",CR,CR) ;
                }

          if ( option == 1 )
                {
                FP = fopen("flagTN", "w") ;
                putc('2', FP) ;             /* Receive WES output.....*/
                fclose(FP) ;

                fdprintf(fout,"%c",e) ;     /* Pipe is on here.....*/
                }                           /* e = CR or escape.....*/
```

```
        if ( option == 2 )
                {
                fpTN = fopen(linkdevice, "w") ;
                if (( e == 012 ) || ( e == 015 ) || ( flag == 0 ))
                        {
                        putc(CR, fpTN) ;
                        putc('@', fpTN) ; /* To simulate the ..........*/
                        printf("@") ;     /*  system prompt  ..........*/
                        }
                fclose(fpTN) ;
                }

        return ;
        }


next()  /* Find the next "matching" macro.......................... ..*/
        {
        int mm ;
        extern int flagN ;
        extern char buffer[81] ;

        for ( j = j; j <= MACSIZE; j = j + 1 )
                {
                if ( mac[j] != d )
                        continue ;

                if ( mac[j] == d )
                        {
                        for (mm = 1; mm <= k; mm = mm + 1 )
                            {
                            if ( mac[j + mm] != buffer[1 + mm] )
                                break ;  /* Go to next matching letter..*/
```

```
                                if ( mac[j + mm] == buffer[1 + mm] )
                                    {
                                    if ( mm == k )
                                            {
                                            return ;
                                            }

                                    else if ( mm < k )
                                            continue ;
                                    }
                                }
                            }
                        }

                flagN = 1 ;

                return ;
                }


xlate() /* Output the WES equivalent of a macro .....*/
                {
                int num ;
                extern int fout ;
                extern int flagCR ;
                extern int option ;
                extern FILE *fpTN ;

                c = getchar() ;
```

```
if (( e != '\n' ) &&
    ( e != 015 )  &&
    ( e != 033 ))
        {
        k = k + 1 ;
        buffer[k + 1] = e ;
        flagCR = 0 ;
        return ;
        }
else if (( e == '\n') || ( e == 015 ) || ( e == 033 ))
        flagCR = 1 ;    /* Then do the following statements ....*/
printf("\n\tSENDING --> ") ;
num = j ;
j = j + 20 ;
fpTN = fopen(linkdevice, "w") ;
for ( j = j; j < num + 21 + SUBMAC ; ++j )
        {
        if ( mac[j] == '$')
                {
                if (( option == 1 ) && ( e == '\n' ))
                    fdprintf(fout,"%c",CR) ; /* Pipe is on here.*/
                if (( option == 1 ) && ( e == 033 ))
                    fdprintf(fout," ") ;     /* Pipe is on here.*/
                break ;
                }
        if ( option == 1 )
                fdprintf(fout,"%c",mac[j]) ; /* Pipe is on here.*/

        if ( option == 2 )
                {
                putc(mac[j], fpTN) ;
                putchar( mac[j] ) ;    /* Echo to user terminal.*/
                }
        }
```

```
        fclose(fpTN) ;

        if ( option == 1 )
                sleep(5) ;

        printf("%c \tRETURNING TO THE STANDARD WFS INPUT MODE. %c",CR,CR);

        return ;
        }

sub()   /* Process a macro requiring substitution.....................*/
        {
        char arg1[400] ;
        int ii ;                /* The index for arg1.....................*/
        int snum ;
        int index ;             /* The index for which arg is being used..*/
        extern int fout ;
        extern int flagCR ;
        extern int option ;
        extern FILE *fpTN ;

        for ( ii = 0; ii <= 399; ++ii )
                arg1[ii] = '\0' ;       /* Clear string "arg1".........*/

        ii = 0 ;
        index = 0 ;     /* = 0,80,160,240,320.........................*/
      subloop:
        c = getchar() ;

        if (( c == 010) || ( c == 020))
                {
                putchar(c) ;
                goto subloop ;
                }
```

```
if ( e == ')' )
        {
        e = getchar() ;
        if (( e == '\n') || ( e == 015 ) || ( e == '[[' ))
                {
                arg1[ii] = '\0' ;
                flagCR = 2 ;
                goto print ;
                }
        else
                {
                err() ;
                return ;
                }
        }

else if ( e == ',' )
        {
        arg1[ii] = '\0' ;
        index = index + 80 ;
        ii = index ;
        }

else
        {
        arg1[ii] = e ;
        ii = ii + 1 ;
        }

goto subloop ;
```

```
print:
  printf("\n\tSENDING --> ") ;
  snum = j ;
  j = j + 20 ;
  fpTN = fopen(linkdevice, "w") ;
  ii = 1 ;
  for ( j = j; j < snum + 21 + SUBMAC ; ++j )
          {
          if ( mac[j] == '$')  break ;
          if ( mac[j] == '[A' )
                  {
                  for(ii = 1;ii <= 79; ++ii)
                      {
                      if ( arg1[ii] == '\0' )
                          break ;
                      if ( option == 1 )
                          fdprintf(fout,"%c",arg1[ii]) ;
                      if ( option == 2 )
                          putc(arg1[ii], fpTN) ;
                          printf("%c",arg1[ii]) ;
                      }
                  }
          if ( mac[j] == '[B' )
                  {
                  for(ii = 80;ii <= 159; ++ii)
                      {
                      if ( arg1[ii] == '\0' )
                          break ;
                      if ( option == 1 )
                          fdprintf(fout,"%c",arg1[ii]) ;
                      if ( option == 2 )
                          putc(arg1[ii], fpTN) ;
                          printf("%c",arg1[ii]) ;
                      }
                  }
```

```
if ( mac[j] == '[C' )
        {
        for(ii = 160;ii <= 239; ++ii)
            {
            if ( arg1[ii] == '\0' )
                break ;
            if ( option == 1 )
                fdprintf(fout,"%c",arg1[ii]) ;
            if ( option == 2 )
                putc(arg1[ii], fpTN) ;
                printf("%c",arg1[ii]) ;
            }
        }

if ( mac[j] == '[D' )
        {
        for(ii = 240;ii <= 319; ++ii)
            {
            if ( arg1[ii] == '\0' )
                break ;
            if ( option == 1 )
                fdprintf(fout,"%c",arg1[ii]) ;
            if ( option == 2 )
                putc(arg1[ii], fpTN) ;
                printf("%c",arg1[ii]) ;
            }
        }
```

```
            if ( mac[j] == '[E' )
                    {
                    for(ii = 320;ii <= 399; ++ii)
                        {
                        if ( arg1[ii] == '\0' )
                            break ;
                        if ( option == 1 )
                            fdprintf(fout,"%c",arg1[ii]) ;
                        if ( option == 2 )
                            putc(arg1[ii], fpTN) ;
                            printf("%c",arg1[ii]) ;
                        }
                    }

            if ( option == 1 )
                    fdprintf(fout,"%c",mac[j]) ; /* Pipe is on here.*/

            if ( option == 2 )
                    {
                    putc(mac[j], fpTN) ;
                    putchar( mac[j] ) ;     /* Echo to user terminal.*/
                    }
            }
    fclose(fpTN) ;
    arg1[0]    = '\0' ;
    arg1[80]   = '\0' ;
    arg1[160]  = '\0' ;
    arg1[240]  = '\0' ;
    arg1[320]  = '\0' ;
    if ( option == 1 )
            sleep(5) ;
    printf("%c %c",CR,CR) ;
    printf("\tRETURNING TO THE STANDARD WES INPUT MODE. %c",CR) ;
    return ;
    }
```

```
edit()  /* Edit a file..................................... */
        {
        char a ;
        char arg[71] ;
        extern int option ;

        if ( option == 1 )
                system("modtty echo") ;

      label4 :
        printf("%c %c \tTYPE IN THE NAME OF THE FILE ",CR,CR) ;
        printf("YOU WANT TO EDIT, ") ;
        printf("%c \t\tFOLLOWED BY A CARRIAGE RETURN ---> ",CR) ;

        for ( i = 0; i <= 70; i = i + 1)
                arg[i] = ' ' ;

        if(termtype == 1)
                {
                arg[0] = 'a' ;
                arg[1] = 'a' ;  /* For Ann Arbor terminals.....*/
                arg[2] = ' ' ;
                arg[3] = ' ' ;
                }

        if(termtype == 2)
                {
                arg[0] = 'v' ;
                arg[1] = 't' ;  /* For Digital VT 100 terminals.....*/
                arg[2] = ' ' ;
                arg[3] = ' ' ;
                }
```

```
if(termtype == 3)
        {
        arg[0] = 'e' ;
        arg[1] = 'd' ;  /* For "dumb" terminals.................*/
        arg[2] = 'n' ;
        arg[3] = ' ' ;
        }

for ( i = 4; i <= 70; i = i + 1)
        {
        arg[i] = getchar() ;     /* Accepts the name of the .....*/
        if (( arg[i] == '\n') ||/*  file you want to edit. .....*/
            ( arg[i] == 015 ))
                {
                arg[i] = '\0' ;
                break ;
                }
        }

printf("%c %c \tEDIT FILE ",CR,CR) ;

for ( i = 3; arg[i] != '\0'; ++i )
        putchar(arg[i]) ;

printf("%c %c \tIS THIS CORRECT ? (TYPE y OR n) --> ",CR,CR) ;

system("modtty brkall") ;
```

```
a = getchar() ;
switch(a)     /* Selection from "y" or "n".....................*/
        {
        case 'y' :
                break ;
        case 'Y' :
                break ;
        case 'n' :
                goto label4 ;
                break ;
        case 'N' :
                goto label4 ;
                break ;
        default :
                err() ;
                goto label4 ;
                break ;
        }

printf("%c %c %c \t\t\t",CR,CR,CR) ;

for ( i = 0; arg[i] != '\0'; i = i + 1 )
        putchar( arg[i] ) ;

printf("%c %c \t\tGOING TO THE EDITOR NOW. %c",CR,CR,CR) ;
printf("%c %c",CR,CR) ;
system(arg) ;

if ( option == 1 )
        system("modtty -echo") ;

return ;
}
```

```
display()   /* List the current filenames.............................*/
        {
        char r ;

        printf("%c%c\tDISPLAYING THE CURRENT LIST OF FILENAMES.",CR,CR) ;
        printf("%c%c",CR,CR) ;
        system("ls | mc") ;
        printf("%c%c",CR,CR) ;
        printf("  TYPE ANY CHARACTER TO RETURN TO THE MENU -->") ;
        r = getchar() ;

        return ;
        }

rvwMacro()   /* Review the current list of macro's.....................*/
        {
        char m3 ;
        FILE *fp1 ;
        printf("%c %c \tEDITING THE MACRO EXPANSION FILE NOW.",CR,CR) ;
        printf("%c %c",CR,CR) ;
        if(termtype == 1)
                system("aa mac.text") ;
        if(termtype == 2)
                system("vt mac.text") ;
        if(termtype == 3)
                system("edn mac.text") ;
        fp1 = fopen("mac.text", "r") ;  /* Read the macro list and the .*/
        i = 0 ;                         /* list of substitutions into  .*/
        while ((m3 = getc(fp1)) != EOF) /*       the array "mac".       .*/
                mac[++i] = m3 ;
        fclose(fp1) ;

        return ;
        }
```

```
orderPrep()  /* Prepare an order for input to WES.....................*/
        {    /*.........NOT A COMPLETE FUNCTION YET.....................*/
        char m4 ;
        FILE *fp4 ;

      label2 :

        fp4 = fopen("orderPrep.text", "r") ;
        while(( m4 = getc(fp4)) != EOF )
                putchar(m4) ;
        fclose(fp4) ;

    /*  m4 = getchar() ; */
        switch(m4) {  /* selection from orderPrep menu.................*/
                case '1' :
                        break ;
                case '2' :
                        break ;
                case '3' :
                        break ;   /* = return in this case..... */
                default :
                        break ;
                   }

    /*  if ( m4 == '1' || m4 == '2' )   */
    /*          goto label2 ;           */

        return ;
        }
```

```
rvwOrders()  /* Review the orders that are already prepared.............*/
        {
        printf("%c%c\tTHE FOLLOWING ARE THE CURRENT ORDER FILES :",CR,CR);
        printf("%c%c",CR,CR) ;
        system("ls *.ord | nc") ;

        return ;
        }

inTops20()   /* Outputs the commands required to telnet and log in......*/
        {
        char m4 ;
        char ttyin[2] ;
        char cmd[23] ;
        FILE *fp7 ;
        extern int fout ;
        extern int popen() ;

        ttyin[0] = '1' ;  /* Write mode for popen().....................*/
        ttyin[1] = '\0' ;
        mode = atoi(ttyin) ;

        cmd[0]  = 't' ;
        cmd[1]  = 'e' ;
        cmd[2]  = 'l' ;
        cmd[3]  = 'n' ;
        cmd[4]  = 'e' ;
        cmd[5]  = 't' ;
        cmd[6]  = ' ' ;          /* telnet tops20 | readTN .............*/
        cmd[7]  = 't' ;
        cmd[8]  = 'o' ;
        cmd[9]  = 'p' ;
        cmd[10] = 's' ;
        cmd[11] = '2' ;
        cmd[12] = '0' ;
```

```
cmd[13] = ' ' ;
cmd[14] = '|' ;
cmd[15] = ' ' ;
cmd[16] = 'r' ;
cmd[17] = 'e' ;          /* telnet tops20 | readTN .............*/
cmd[18] = 'a' ;
cmd[19] = 'd' ;
cmd[20] = 'T' ;
cmd[21] = 'N' ;
cmd[22] = '\0' ;
printf("%c\tCOMPLETED CALL TO OPEN TELNET - PLEASE WAIT.%c",CR,CR);
printf("\t--------- ---- -- ---- ------   ******* **** %c",CR) ;
fout = popen(cmd,mode) ;
sleep(3) ;
system("modtty crlfout pagelen 0 flowout xtabs nl3") ;

fp7 = fopen("tnTops20.text", "r") ;
while ((m4 = getc(fp7)) != ECF)
        {
        fdprintf(fout,"%c",m4) ;   /* Pipe is ON here.....*/
        if (( m4 == '\n' ) || ( m4 == 015 ))
                sleep(2) ;
        }
fclose(fp7) ;

system("modtty crlfout pagelen 0 flowout xtabs nl3") ;
sleep(4) ;
printf("%c %c",CR,CR) ;
printf("\tRETURNING CONTROL OF THE KEYBOARD TO YOU.") ;
printf("%c",CR) ;
printf("\t      (THE STANDARD WES INPUT MODE)") ;
printf("%c",CR) ;
fdprintf(fout,"%c",CR) ;  /* Pipe is on here.....*/
return ;
}
```

```
c3LabLink()     /* Link to another terminal in the C-3 Lab.............*/
        {
        char b ;
        char m5 ;
        char dev2[4] ;
        extern char device[12] ;
        extern FILE *fpTN ;
        FILE *fp8 ;

        fp8 = fopen("c3LL.text", "r") ;  /* Displays the list of .......*/
        while ((m5 = getc(fp8)) != EOF)  /*   possible links.    .......*/
                putchar(m5) ;
        fclose(fp8) ;

      label6:
        printf("\n\t\t\t----> tty") ;
        device[8] = getchar() ;  /* Accepts the tty name...............*/
        printf("%c%c\tLINK TO ",CR,CR) ;

        for ( i = 5; i <= 8; ++i )
                putchar(device[i]) ;

        printf("%c%c\tIS THIS CORRECT ? (TYPE y OR n) --> ",CR,CR) ;

        system("modtty brkall") ;
        b = getchar() ;
        switch(b)    /* Selection from "y" or "n"....................*/
                {
                case 'y' :
                        break ;
                case 'Y' :
                        break ;
                case 'n' :
                        goto label6 ;
                        break ;
```

```
                case 'N' :
                        goto label6 ;
                        break ;
                default :
                        err() ;
                        goto label6 ;
                        break ;
                }

        for ( i = 0; i <= 3; ++i )
                dev2[i] = device[i + 5] ;

        printf("%c%c\tLINKING TO C-3 LAB TERMINAL %.4s.%c",CR,CR,dev2,CR);
        printf("%c%c\tRETURNING CONTROL OF THE KEYBOARD TO YOU.",CR,CR) ;
        printf("%c@",CR) ;

        linkdevice[8] = dev2[3] ;

        fpTN = fopen(linkdevice, "w") ;
        putc(CR, fpTN) ;          /* To simulate the .....................*/
        putc('@', fpTN) ;         /*  system prompt  .....................*/
        fclose(fpTN) ;

        return ;
        }


help()  /* The help routine.....................................*/
        {
        char m6 ;
        extern int fout ;
        extern int option ;
        FILE *fp2 ;
        extern FILE *FP ;
```

```
        fp2 = fopen("help.text", "r") ;
        while ((m6 = getc(fp2)) != EOF)
                putchar(m6) ;
        fclose(fp2) ;

        if ( option == 1 )
                {
                FP = fopen("flagTN", "w") ;
                putc('2', FP) ;              /* Receive WES output.....*/
                fclose(FP) ;

                fdprintf(fout,"%c",CR) ;  /* Pipe is on here.....*/
                }

        if ( option == 2 )
                {
                fpTN = fopen(linkdevice, "w") ;
                putc(CR, fpTN) ;
                putc('@', fpTN) ; /* To simulate the .....*/
                printf("@") ;     /*  system prompt  .....*/
                fclose(fpTN) ;
                }

        return ;
        }


err()   /* If the option chosen does not exist.....*/
        {
        extern int option ;
        extern FILE *FP ;

        printf("%c \t*** NOT A VALID INPUT.*** TRY AGAIN.*** %c",CR,CR) ;
        printf("%c %c",CR,CR) ;
```

```
        if ( option == 1 )
                {
                FP = fopen("flagTN", "w") ;
                putc('2', FP) ;          /* Receive WES output.....*/
                fclose(FP) ;
                }

        return ;
        }


quit()  /* To exit from the program, type "$quit".....*/
        {
        extern int fout ;
        extern int pclose() ;

        printf("%c%c\tTHE PROGRAM IS TERMINATING GRACEFULLY.%c",CR,CR,CR);
        printf("%c%c",CR,CR) ;
        pclose(fout) ;
        system("modtty normal 9600") ;
        exit() ;
        }


popen(pcmd,pmode)       /* Open pipe file descriptor (fd).............*/

#define test(a,b) (mode == READ ? (b) : (a))
        char *pcmd ;
        int pmode ;
        {
        int p[2] ;
        extern int popen_pid ;

        if (pipe(p) < 0)        /* Create a pipe w/ a system call.....*/
                return(NULL) ;
```

```
        if ((popen_pid = fork()) == 0)  /* Fork to create two copies....*/
                {
                close(test(p[WRITE],p[READ])) ; /* Close unused side....*/
                close(test(0,1)) ;      /* Close stdin and stdout.......*/
                dup(test(p[READ],p[WRITE])) ;   /* Pipe = stdin/out.....*/
                close(test(p[READ],p[WRITE])) ; /* Close pipe = stdin...*/
                execl("/bin/sh","sh","-c",pcmd,0) ;
                exit(1) ;
                }
        if (popen_pid == -1)
                return(NULL) ;
        close(test(p[READ],p[WRITE])) ;
        return(test(p[WRITE],p[READ])) ;
        }

pclose(fd)      /*Close pipe file descriptor (fd) ..........................*/
        int fd ;
        {
        register r,(*hstat)(),(*istat)(),(*qstat)() ;
        int status ;
        extern int popen_pid ;
        close(fd) ;
        istat = signal(SIGINT,SIG_IGN) ;
        qstat = signal(SIGQUIT,SIG_IGN) ;
        hstat = signal(SIGHUP,SIG_IGN) ;
        while((r = wait(&status)) != popen_pid && r != -1) ;
                if (r == -1)
                        status = -1 ;
        signal(SIGINT,istat) ;
        signal(SIGQUIT,qstat) ;
        signal(SIGHUP,hstat) ;
        return(status) ;
        }

/*.........................END OF SUBROUTINES.....................*/
```

APPENDIX C

THE MAIN PROGRAM TEXT FILES

FILE "mac.text"

MACRO LIST	STRING TO BE SUBSTITUTED FOR THE MACRO
cap(s,n)$	ENTER PLAN C[B FOR [A LAUNCH 4 F14 [B Ø3Ø 24Ø 28ØØØ LOAD 4 PHENX 4 SWDR MISSION CAP STOP $
atk.a7(s,#,n)$	FOR [A LAUNCH [B A7E [C 12Ø 24Ø 1ØØØ $
radar(s)$	FOR [A ACTIVATE RADAR $
macro5$	submac5 $
macro6$	submac6 $
macro7$	submac7 $
macro8$	submac8 $
mac9(x)$	Firstword is [A. $
mac10(x,y)$	Firstword [A, secondword [B, firstagain [A. $

THE LAST CHARACTER OF BOTH THE MACRO AND ITS EXPANSION
MUST BE THE DOLLAR SIGN CHARACTER "$"
==== ==

(THE "[" ABOVE INDICATES A CONTROL CHARACTER)

s = ship # = quantity n = flight name

FILE help.text

"$ help" or "$?" causes this file to be displayed.

The "$" character is used to enter the portion of the program which is NOT transparent to you. The following syntax must be followed:

1. "$ CR" ($ followed by a carriage return) -

   Used to enter the 'Menu of Options' mode.

2. "$ macro CR" ($ followed by a macro AND a carriage return) - (OR an escape character)

   Used to expand a macro into a WES command (or into a series of WES commands).

3. "$ quit CR" ($ followed by quit AND a carriage return) -

   Used to exit the program gracefully.

*** RETURNING TO THE STANDARD WES INPUT MODE.***

FILE open.text

WELCOME - You have the following options:

1. Telnet to Tops20 and log in automatically.
2. Link to another terminal in the C-3 Lab
   (primarily for training on this program).

*** TYPE IN 1 OR 2.***

FILE menu.text

MENU #1 - You have the following options:

1. Review/Edit an existing file.
2. See the current list of filenames.

3. Review/Edit the current macro list.

4. Prepare an order.
5. See the current list of orders.

6. Quit (return to standard WES input mode).

*** Type in 1, 2, 3, 4, 5, or 6. ***

FILE orderPrep.text

ORDER PREPARATION - NOT IMPLEMENTED YET.

FILE tnTops20.text

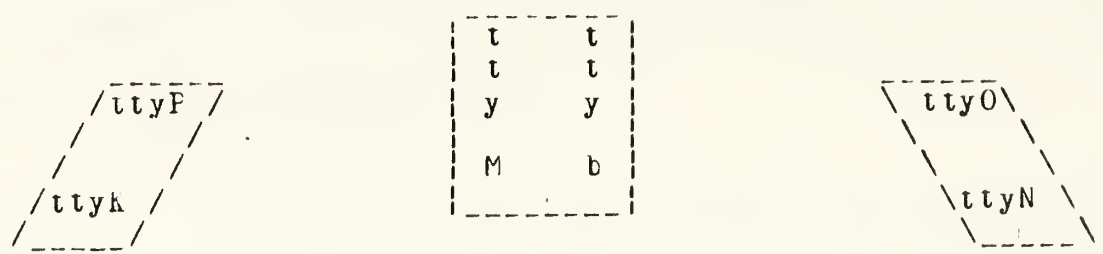
log nps12 password
(OTHER COMMANDS HERE)

FILE c3LL.text

Choose one of the following devices to link to :

tty	ttyA	ttyB	ttyC	ttyD
ttyE	ttyF	ttyG	ttyH	ttyJ
ttyK	ttyL	ttyM	ttyN	ttyO
ttyP	ttyQ	ttyR	ttyS	ttyT
ttyU	ttyV	ttyW	ttyX	ttyY
ttya	ttyb	ttyc	ttyd	ttye
ttyf	ttyg	ttyh	ttyi	ttyj
ttyk	ttyl	ttym	ttyn	ttyo
ttyp	ttyw	ttyy	ttyz	

The current tty layout in the lab is :
(Ann Arbor Terminals)



*** TYPE IN THE tty YOU WANT TO LINK TO ***
*** FOLLOWED BY A CARRIAGE RETURN ***
*** FOR EXAMPLE ----> ttyL ***

APPENDIX D

THE WES OUTPUT BUFFER PROGRAM

```
#
#include <stdio.h>

                /* COMPILE THIS USING "pcc readTN.c -lS" */

char a,                 /* The "flag" character.............................*/
     b,                 /* Characters from telnet...........................*/
     buffer[6400],      /* The WES output buffer............................*/
     *fp2 ;             /* The pointer to the flag file.....................*/
int  flag,              /* Equal to 1 if the buffer has been filled.........*/
     i, j, k,           /* Index integers...................................*/
     savei ;            /* The beginning of the buffered output if .........*/
                        /*  the buffer has been filled......................*/
main()  /* readTN.c ...................................................*/
        {
        for ( j = 0; j <= 6400; ++j )
                buffer[j] = ' ' ;

        i = 0 ;         /* Current position in buffer.....................*/
        a = '1' ;       /* Buffer output when program first begins........*/
        flag = 0 ;      /* Buffer is NOT full.............................*/
      loop:
        b = getchar() ;
        if ( b == EOF )
                exit() ;
        if ( a == '1' )         /* Buffer WES output.....................*/
                buff() ;
        if ( a == '2' )         /* Send WES output.......................*/
                putchar(b) ;
```

```
        if (( b == 012) || ( b == 015)) /* When at the end of a line.....*/
                {                       /*   check the buffer flag  .....*/
                fp2 = fopen("flagTN", "r") ;
                a = getc(fp2) ;
                fclose(fp2) ;

                if ( a == '1' )   /* Buffer WES output..................*/
                        buff() ;
                }
        goto loop ;
        }
buff()  /* Buffer output when  a = '1' ................................*/
        {
        buffer[i] = b ;
        if (( b == 012) || ( b == 015))
                {
                fp2 = fopen("flagTN", "r") ;
                a = getc(fp2) ;
                fclose(fp2) ;
                if ( a == '2' )     /* Send WES output.................*/
                        {
                        send() ;
                        return ;
                        }
                }
        i = i + 1 ;
        if ( i == 6400 )
                {
                flag = 1 ;     /* Buffer has been filled..................*/
                savei = i ;    /* Keep track of output in the buffer......*/
                i = 0 ;        /* Begin overwriting buffer................*/
                }
        return ;
        }
```

```
send()  /* Send output when a = '2' */
        {
        if ( flag == 0)     /* Buffer has NOT been filled.................*/
                {
                for ( k = 0; k <= i; ++k )
                        {
                        if(( buffer[k] == '\n') && ( buffer[k+1] == '\n'))
                                k = k + 1 ;

                        putchar(buffer[k]) ;
                        }
                }
        if ( flag == 1)     /* Buffer has been overfilled.................*/
                {
                printf("\n\t*** WARNING - BUFFER OVERFLOW ***\n") ;
                printf("\n\t*** SOME OUTPUT HAS BEEN LOST ***\n\n") ;
                for ( k = savei; k <= 6400; ++k ) /* First part of output*/
                        {
                        if (( buffer[k] == '\n') && ( buffer[k+1] == '\n'))
                                k = k + 1 ;
                        putchar(buffer[k]) ;
                        }
                for ( k = 0; k <= i; ++k )  /* Overflow portion of output*/
                        {
                        if (( buffer[k] == '\n') && ( buffer[k+1] == '\n'))
                                k = k + 1 ;
                        putchar(buffer[k]) ;
                        }
                }
        i = 0 ;         /* Go to the beginning of the buffer..............*/
        flag = 0 ;      /* Buffer has been flushed........................*/
        return ;
        }
```

BIBLIOGRAPHY

Kernighan, B. W. and Ritchie, D. M., The C Programming Language, Prentice Hall, 1978.

Programmers Manual for the Unix Operating System, 6th ed, Bolt, Beranek and Newman, 1980.

UNIX Programming - Second Edition, Bell Laboratories, 1978.

Warfare Environment Simulator (WES) User Manual Version 3.1, Naval Ocean Systems Center, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Superintendent ATTN: Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Superintendent ATTN: Chairman, Code 39 C3 Academic Group Naval Postgraduate School Monterey, California 93940	1
4.	Superintendent ATTN: C3 Curriculum Office, Code 39 Naval Postgraduate School Monterey, California 93940	1
5.	Professor John M. Wozencraft, Code 62Wz Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	3

197634

Thesis
F66245 Fotheringham
c.1 An input/output filter program for the warfare environment simulator (WES).